Pipelined Implementation

of

Densely Packed Decimal Encoding

A Thesis Submitted in partial fulfilment of the requirements for the Degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Kamlesh Sulanki

Department of Computer Science and Engineering

National Institute of Technology

Rourkela

May 2011

National Institute of Technology, Rourkela

Certificate

This is to certify that the work in the thesis entitled **Pipelined Implementation of Densely Packed Decimal Encoding** submitted by Kamlesh Sulanki is a record of an authentic work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering at National Institute of Technology, Rourkela.

Dr. Ashok Kumar Turuk
Associate Professor
Head of the Department
Computer Science and Engineering
NIT Rourkela

Place : NIT Rourkela
Date :

Acknowledgement

I would like to express my sincere gratitude to my supervisor **Dr. Ashok Kumar Turuk**, Department of Computer Science and Engineering, NIT Rourkela, for his guidance, help, suggestions, never ending support and faith, without which this thesis and my work would not be have been possible.

I would also like to thank the senior members of Embedded Systems and VLSI lab of the *Electronics Department* who provided me with all the required equipments and facilities to carry out my hardware simulation work successfully.

I would also like to express my gratitude and love to my family and friends for their support and their helping hands which were always extended towards me.

Abstract

The BCD (Binary Coded Decimal) is one of the most popular encoding scheme for decimal numbers in which each digit is represented by its own binary sequence. A decimal digit (base 10) , 0 through 9, can be can be represented as a sequence of 4 bits in BCD encoding scheme. However, when representing decimal numers, 0 through 9, only 10 out of 16 possible binary sequences are used. Tien Chi Chen and Irving T. Ho proposed an encoding scheme in 1975 now known as Chen-Ho encoding.It encodes three decimal digits in 10 bits (which require 12 bits in BCD encoding scheme) using an algorithm which can be applied or reversed using only simple Boolean operations, but limited to the fact that number of digits should be a multiple of 3. An improvement to the encoding which has the same advantages but is not limited to multiples of three digits was described by M.F. Cowlishaw called Densely Packed Decimal (DPD) encoding allows arbitrary-length decimal numbers to be coded efficiently. The IEEE-754-2008 Packed Decimal Encoding (PDE) is a way of encoding decimal numbers. The core of the packed-decimal encoding IEEE-754-2008 is the DPD encoding sceme. The DPD encoding (BCD to DPD) and the decoding (DPD to BCD)mechanism must be very fast as it is applied for every decimal number in every calculation. Moreover, DPD encoding can also be used in data communication which can help in reducing the number of bits to be transmitted or received. This thesis includes the work for a pipelined implementation of a DPD encoder and decoder, its simulation on hardware using VHDL and Xilinx Spartan 3E FPGA.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Most modern computer systems are equipped with a floating point unit or accelator. Representation of real numbers using a fixed number of bits or memory and their manipulation with high degree of precision and speed has always been an area of interest for scientific research and has become crucial in commercial and financial applications. Due to the growing importance and need for efficient decimal floating-point arithmetic, the older IEEE 754-1985 standard was replaced by the new IEEE 754-2008 standard which can represent decimal fractions exactly and perform decimal rounding more efficiently.

The IEEE 754-2008 standard includes almost all of the original IEEE 754-1985 standard and the IEEE 754-1987 standard for radix independent floating point arithmetic, which allows the radix for representation to be 2 or 10. The binary encoding (radix 2) allows for efficient software operations, using the native binary integer operations of a processor. The decimal encoding (radix 10), also known as densely packed decimal or DPD, was designed to make a hardware implementation of decimal floating-point arithmetic as efficient as possible [4]. In the DPD format, the significand is encoded in a group of 3 digits, each group encoded in 10 bits (declets). It was proposed by M.F. Cowlishaw and is a refinement of Chen-Ho encoding scheme[2].

1.2 Motivation

It is said that 'Hardware is always faster than software.' Hardware implementations of decimal floating-point arithmetic operations are one to two orders of magnitude faster than software implementations [7][8]. The growing importance of decimal floating point arithmetic and its ubiquitousness has evoked a need for hardware units for its support. BCD has remained a popular choice for representation of decimal numbers in computer systems. But, as pointed out earlier, DPD encodes three decimal digits into 10 bits which require 12 bits if BCD representation is used. The DPD encoder and decoder hardware units must provide a fast conversion from BCD to DPD and back from DPD to BCD. Moreover, DPD scheme can be used in data communication and transmission only if the underlying hardware is fast.

Chapter 2

Literature Review

2.1 Binary Coded Decimal(BCD) Encoding

The Binary Coded Decimal (BCD) notation is one of the most simple way of notating a decimal number. This notation requires a four bit unit for each digit to store the ten values 0 to 9. The BCD is quite concise and simple. The four-bits unit wherein a BCD-digit is stored is called a 'nibble'.

❖ BCD-storage:

Digit	Bit-pattern
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Figure 2.1: BCD encoding

Thus, the BCD encoding for the number 127 would be: 0001 0010 0111 Whereas the pure binary number would be: 0111 1111

BCD encoding is capable of representing six more values, 10 to 15, but these values are never used. This means that a lot of 'value space' (or: 'bit-pattern space' = set of available bit-patterns) is wasted, 37.5 percent for this digit. For example: A sequence of twelve bits can contain 4096 different bit patterns. When three digits are stored in this sequence only 1000 of these patterns are used. The other 3096 patterns are discarded. This leads to wastage of 75.6 percent.

Therefore a compression to save the value space must be employed. For example, a sequence of ten bits

can contain 1024 different bit patterns. This is slightly more than 1000, the minimum number required to store three digits. So a compression of the twelve into the ten bits is possible. This compression reduces the wastage from 75.6 percent to only 2.4 percent.
If a decimal digit requires four bits, then three decimal digits require 12 bits. However, since $2^{10} > 10^3$, if three decimal digits are encoded together then only 10 bits are needed. Two such encodings are Chen-Ho encoding [1] and Densely Packed Decimal [2]. The latter has the advantage that subsets of the encoding encode two digits in the optimal 7 bits and one digit in 4 bits, as in regular BCD.

2.2 Chen-Ho Encoding

The Chen-Ho encoding[1] allows three decimal digits to be represented in ten binary bits, which may have up to 1,024 possible different values, and which therefore can encode the 1,000 possibilities for three digits with only a little waste. The advantage that is incurred from Chen-Ho encoding over a straightforward binary representation in 10 bits is that only simple Boolean operations are needed for conversion to or from BCD; multiplications and divisions are not required. This encoding also has the advantage over variable length schemes, because its fixed-length mapping allows simpler encoding and decoding in either hardware or software. The Chen-Ho scheme works very well when decimal numbers have lengths which are multiples of three decimal digits, as this encoding packs three digits into 10 bits with little waste. It is less satisfactory for other lengths [2].

2.3 Densely Packed Decimal(DPD) Encoding

Densely Packed Decimal Encoding proposed by M. F. Cowlishaw [2] is an improvement over Chen-Ho encoding. It uses the coding scheme equivalent to the Chen-Ho but instead of using Huffman-code it uses a fresh arrangement of bits that gives it further advantages over the Chen-Ho scheme. The advantages can be listed as follows:[2]

1. The encoding of decimal digits, unlike Chen-Ho encoding, does not require the number of decimal digits to be a multiple of three. Thus, it can encode arbitrary number of decimal digits. One or two decimal digits are compressed into the optimal four or seven bits respectively.

2. The encoded decimal numbers can be expanded into a longer field simply by padding with zero bits; re-encoding is not necessary. While Chen-Ho encoding requires a re-encoding instead of simple padding if an encoded two digits is expanded into three digit field.

3. When numbers in the range 0 through 79 are encoded by this scheme they have the same right-aligned encoding as in BCD. While in Chen-Ho encoding only the numbers 0 through 7 remains same as in BCD.

These advantages make the new DPD encoding a better choice than Chen-Ho encoding for both hardware and software representations of decimal numbers. Here are some examples of encoding in BCD , Chen-Ho [1] and Densely Packed Decimal [2]:

Decimal	BCD	Chen-Ho	Densely Packed
005	0000 0000 0101	000 000 0101	000 000 0101
009	0000 0000 1001	110 000 0001	000 000 1001
055	0000 0101 0101	000 010 1101	000 101 0101
099	0000 1001 1001	111 000 1001	000 101 1111
555	0101 0101 0101	010 110 1101	101 101 0101
999	1001 1001 1001	111 111 1001	001 111 1111

Figure 2.2: Examples of BCD, Chen-Ho and DPD Encoding

2.3.1 Details of encoding

The DPD encoding, categorizes each of the three digits as follows: Small (0-7, requiring 3 bits) Large (8 or 9, requiring one bit). The most significant bit of each BCD digit is 0 for small values, and 1 for the large values [Figure 2.1]. Encoding/Compression (12 bits to 10 bits) is done as in Figure 2.3. Decoding/Expansion (10 bits to 12 bits) is done as in Figure 2.4.

- 3 digits represented in BCD as abcd, efgh, ijkm
- 3 digits represented in DPD as pqr stu v w

a e i	p q r	s t u	v	w x y
000	bcd	f g h	0	j km
001	bcd	f g h	1	00m
010	bcd	j k h	1	01m
100	j kd	f g h	1	10m
110	j kd	00 h	1	11m
101	f gd	01 h	1	11m
011	bcd	10 h	1	11m
111	00d	11 h	1	11m

Figure 2.3: Encoding/Compression

vwxst	abcd	efgh	ijkm
0	0pqr	0stu	0wxy
100 . .	0pqr	0stu	100y
101 . .	0pqr	100u	0sty
110 . .	100r	0stu	0pqy
11100	100r	100u	0pqy
11101	100r	0pqu	100y
11110	0pqr	100u	100y
11111	100r	100u	100y

Figure 2.4: Decoding/Expansion

2.3.2 Hardware Logic

As proposed by M.F Cowlishaw [2], The encoding is done as in Figure 2.5.

```
p := (b & ~a) | (j & a & ~i) | (f & a & ~e & i)
q := (c & ~a) | (k & a & ~i) | (g & a & ~e & i)
r := d
s := (f &~e &~(a & i)) | (j &(~a & e &~ i)) | (e & i)
t := (g &~e &~(a & i)) | (k & (~a & e &~ i)) | (a & i)
u := h
v := a | e | i
w := a | (e & i) | (j & ~e &~i)
x := e | (a & i) | (k & ~a &~i)
y := m
```

Figure 2.5: Decoding/Expansion

The decoding is done as in Figure 2.6.

```
a := (v & w) & (~x |~s |(s & t))
b := p & (~v j ~w | (x & s & ~t))
c := q & (~v j ~w | (x & s & ~t))
d := r
e := v & ((~w & x) | (w & x & (s j ~t)))
f := (s &(~v |(v &~x))) | (p & v & w & x &~s & t)
g := (t &(~v |(v &~x))) | (q & v & w & x &~s & t)
h := u
i := v &((~w &~x) |(w & x &(s j t)))
j := (w & ~v)| (s & v & ~w & x) | (p & v & w &
      (~x |(~s &~t)))
k := (x & ~v) | (t & v & ~w & x)| (q & v & w &
      (~x |(~s &~t)))
m :=y
```

Figure 2.6: Decoding/Expansion

Chapter 3

Hardware Platform and Simulation

3.1 FPGAs

A field-programmable gate array (FPGA) is an programmable integrated circuit designed to be configured by the customer or designer. A FPGA can be configured to simulate any digital logic or function and can be configured/programmed as many number of times as required. The FPGA configuration is generally specified using a hardware description language (HDL) and a set of tools provided by the FPGA manufacturer.

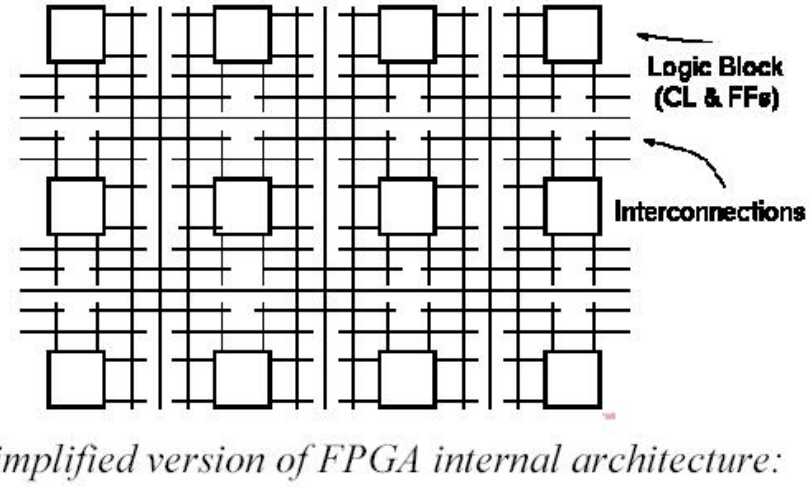

Figure 3.1: FPGA overview [13]

There are (at least) 5 companies making FPGAs in the world. The first two (Xilinx and Altera) hold the bulk of the market.

* Xilinx invented the FPGA and is the biggest name in the FPGA world.

* Altera is the second FPGA heavyweight, also a well-known name.

* Lattice, Actel, Quicklogic and SiliconBlue are smaller players.

FPGAs are built from one basic "logic-cell", duplicated hundreds or thousands of time. A logic-cell is basically a small lookup table ("LUT"), a D-flipflop and a 2-to-1 mux (to bypass the flipflop

if desired). The LUT is like a small RAM that can implement any logic function. It has typically a few inputs like 3 to 6, etc. Each logic-cell can be connected to other logic-cells through interconnect resources (local and global tracking bus / programmable switching matrices placed around the logic-cells). Each cell(Configurable/programmable Logic Block, CLB) can do little, but with lots of them connected together, complex logic functions can be created [12].

3.2 Configuring the FPGA

HDL design entry

High-level HDLs are nowadays the preferred way to create FPGA designs. They also make migrations much easier. VHDL Verilog are the most popular hardware description languages to program the FPGAs. Here, we use VHDL to create a Compression and Expansion Module.
Compression Module has the following basic logic:

p <= ((NOT a) AND b) OR (a AND j AND (NOT i)) OR (a AND f AND i AND (NOT e));
q<= ((NOT a) AND c) OR (a AND k AND (NOT i)) OR (a AND g AND i AND (NOT e));
r<= d;
s<= ((NOT e) AND f AND (NOT(a AND i))) OR ((NOT a) AND (NOT i) AND e AND j) OR (e AND i);
t<= ((NOT e) AND g AND (NOT (a AND i))) OR ((NOT a) AND (NOT i) AND e AND k) OR (a AND i);
u<= h;
v<= a OR e OR i;
w<= a OR (e AND i) OR ((NOT e) AND j AND (NOT i));
x<= e OR (a AND i) OR ((NOT a) AND k AND (NOT i));
y<= m;

where a,b,c,d,e,f,g,h,I,j,k,m represents 12 bits of input BCD number and p,q,r,s,t,u,v,w,x,y signifies 10 bits of output DPD decocded number.
Expansion Module has the following basic logic:
a<= (v AND w) AND ((NOT x) OR (NOT s) OR (s AND t));
b<= p AND ((NOT v) OR (NOT w) OR (s AND (NOT t) AND x));
c<= q AND ((NOT v) OR (NOT w) OR (s AND (NOT t) AND x));
d<= r;
e<= v AND (((NOT w) AND x) OR (((NOT t) OR s) AND w AND x));
f<=(s AND ((NOT v) OR ((NOT x) AND v))) OR (p AND (NOT s) AND t AND v AND w AND x);
g<= (t AND ((NOT v) OR ((NOT x) AND v))) OR (q AND (NOT s) AND t AND v AND w AND x);
h<= u;
i<= v AND (((NOT w) AND (NOT x)) OR (w AND x AND (s OR t)));

j<= ((NOT v) AND w) OR (s AND v AND (NOT w) AND x) OR (p AND v AND w AND ((NOT x)OR ((NOT s) AND (NOT t))));
k<= ((NOT v) AND x) OR (t AND v AND (NOT w) AND x) OR (q AND v AND w AND ((NOT x)OR ((NOT s) AND (NOT t))));
m<= y;

Synthesis

Synthesis takes the design (HDL or schematic) and creates a netlist out of it. A netlist is a "list of nets", connecting basic gates or flipflops together. On synthesizing the compression and expansion module we have the following generated RTL schematics as shown in figure 3.2 and 3.3.

Mapping

Xilinx provides the users with a feature to map a selected I/O line onto a desired LUT or other ports on the FPGA. This can be done by entering the required values in a UCF(User Constraints File) which consists of name of the net and the corresponding LUT or port on the FPGA where we intend to map it. The values of all the available LUTs are supplied in the data-sheet of the hardware.

Place-and-route

Place-and-route describes several processes where the netlist elements are physically places and mapped to the FPGA physical resources, to create a file that can be downloaded in the FPGA chip. Place-and-route is always done via FPGA software from the FPGA vendor [12].

3.3 Simulation

Testbench modules were developed in VHDL for Compressor and Expansion modules with input as decimal 099 or BCD 0000 1001 1001 and corresponding DPD decelet 000 101 1111. Simulation was done using Xilinx ISE 12.1 and ISim. Results were as in Figure 10 and Figure 11.

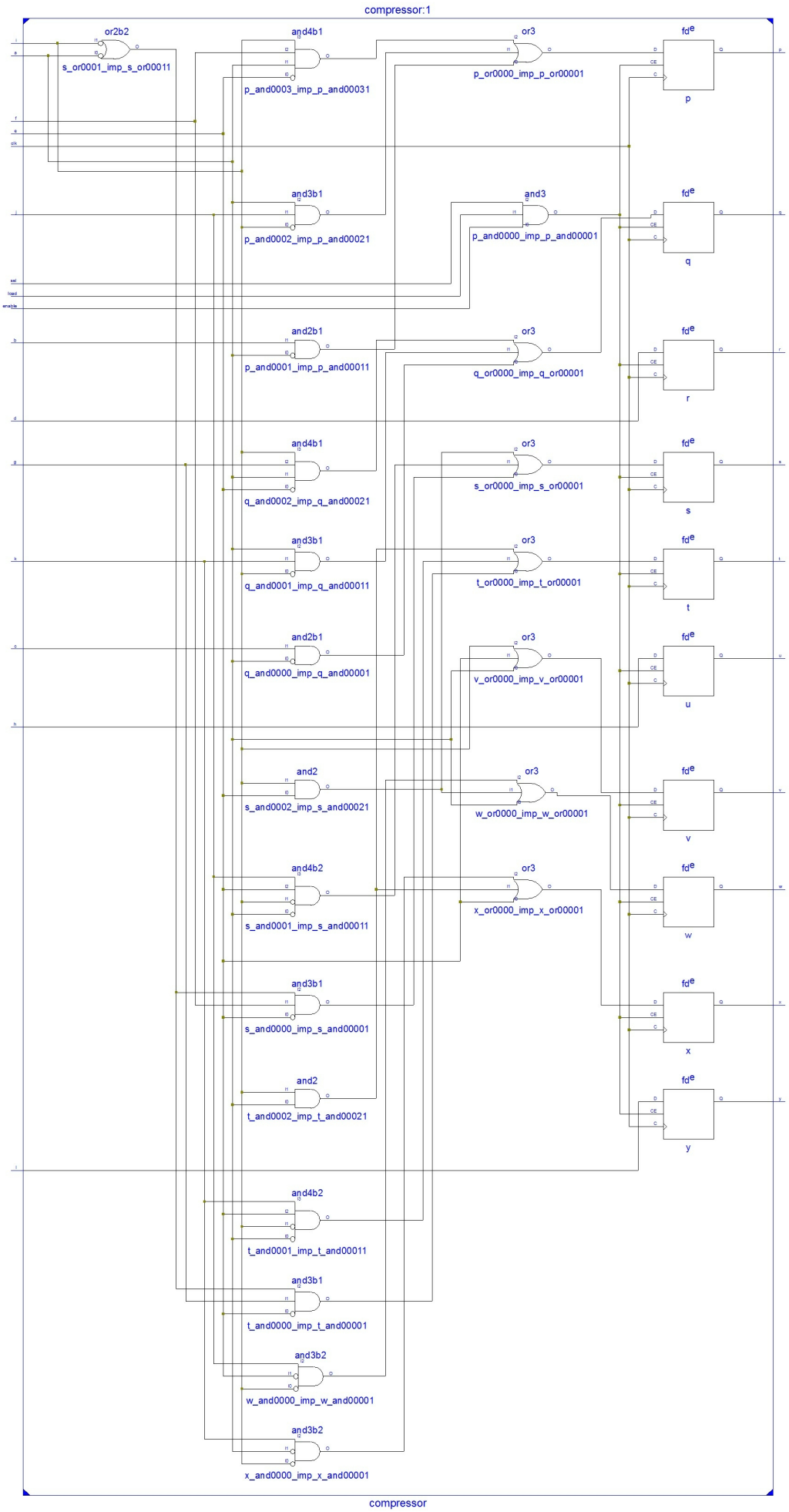

Figure 3.2: RTl schematic for Compressor module

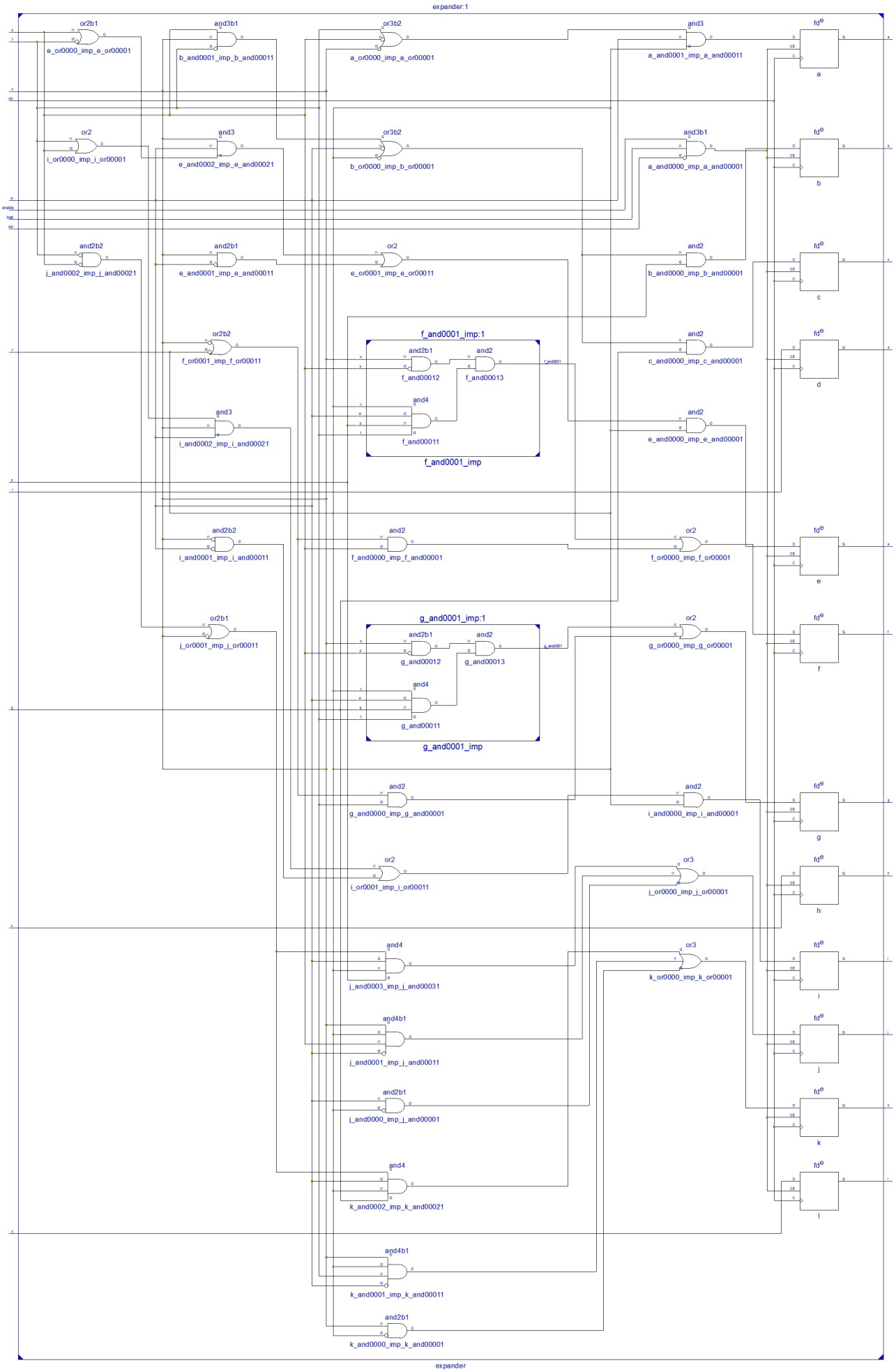

Figure 3.3: RTl schematic for Expansion module

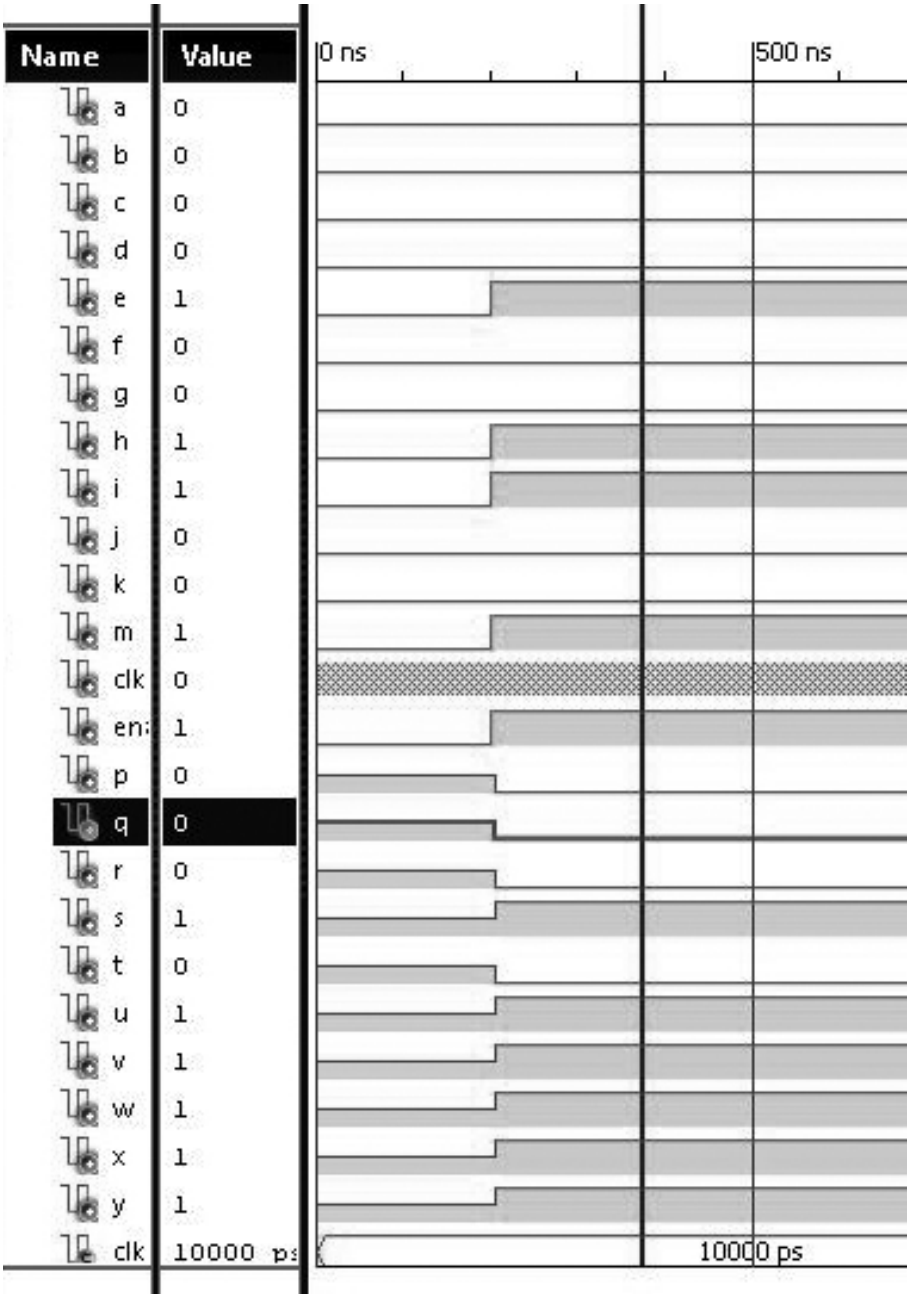

Figure 3.4: Simulation of Compressor module for decimal 099

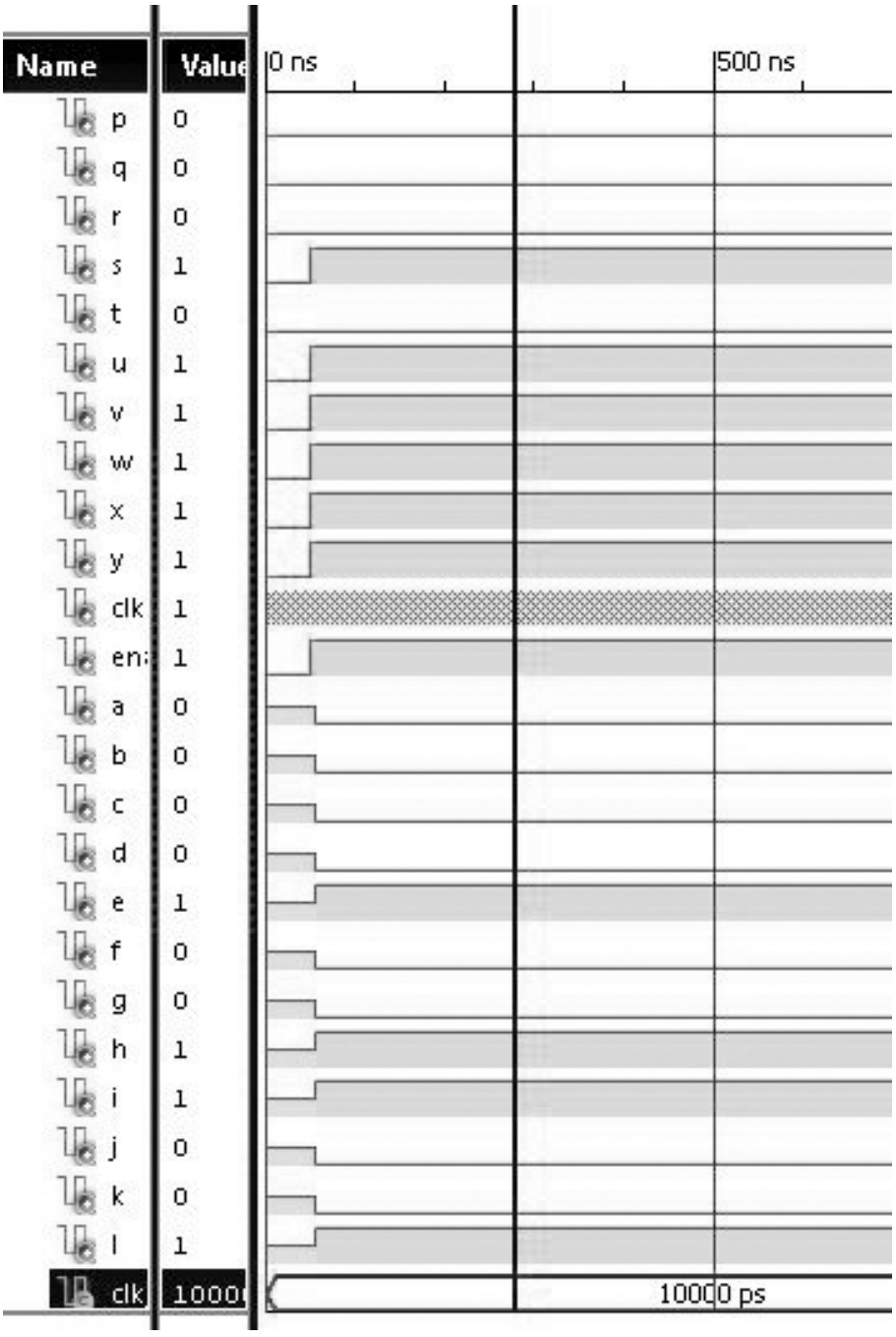

Figure 3.5: Simulation of Expansion module for decimal 099

Chapter 4

Compressor and Expander Module Optimization

4.1 Compressor Module Optimization

We observed that RTL schematic generated by the VHDL code for the compresssor logic module has 4 logic gate level delay(including the not gates represented by a circle in the RTL schematic). But both of these conversions (BCD to DPD and DPD to BCD) can be achieved in three logic gate delays [3]. *Three logic gate delay* module can be achieved by studying each of the output bits or signals separately.

4.1.1 Logic module for signal p

a e i	p
000	b
001	b
010	b
100	j
110	j
101	f
011	b
111	0

Figure 4.1: Logic for signal p

Output signal p can be modelled as follows:

t1 <= b and not a;

t2 <= j and a and not i;

t3 <= f and a and not e and i;

p <= t1 or t2 or t3 ;

Then we have the following logic module as shown in figure 4.2

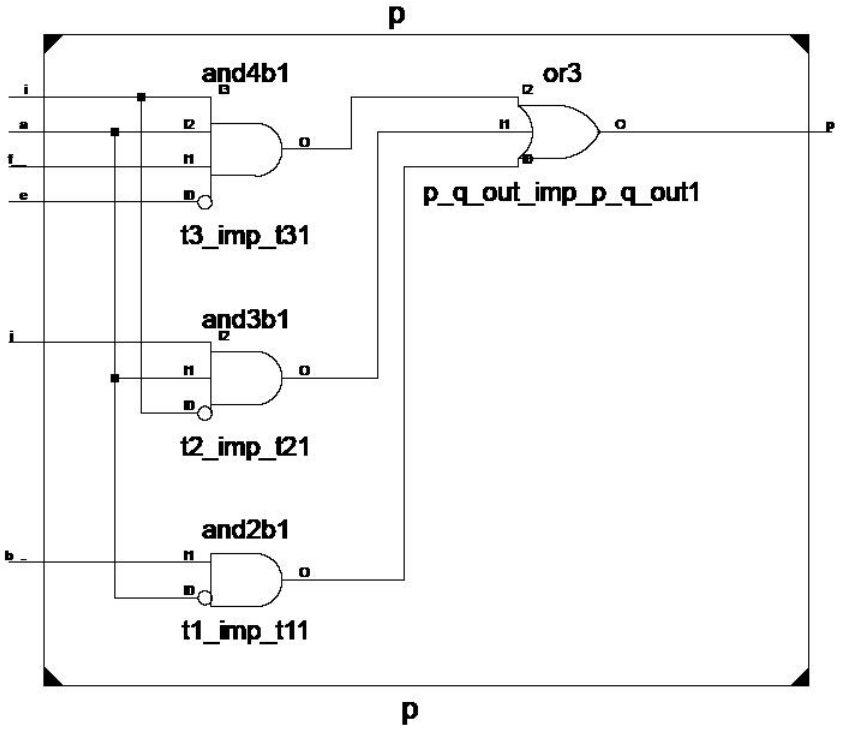

Figure 4.2: Logic module for signal p

4.1.2 Logic module for signal q

a e i	q
000	c
001	c
010	c
100	k
110	k
101	g
011	c
111	0

Figure 4.3: Logic for signal q

Output signal q can be modelled as follows:

t1 <= b and not a;

t2 <= k and a and not i;

t3 <= g and a and not e and i;

q <= t1 or t2 or t3 ;

Then we have the following logic module as shown in figure 4.4

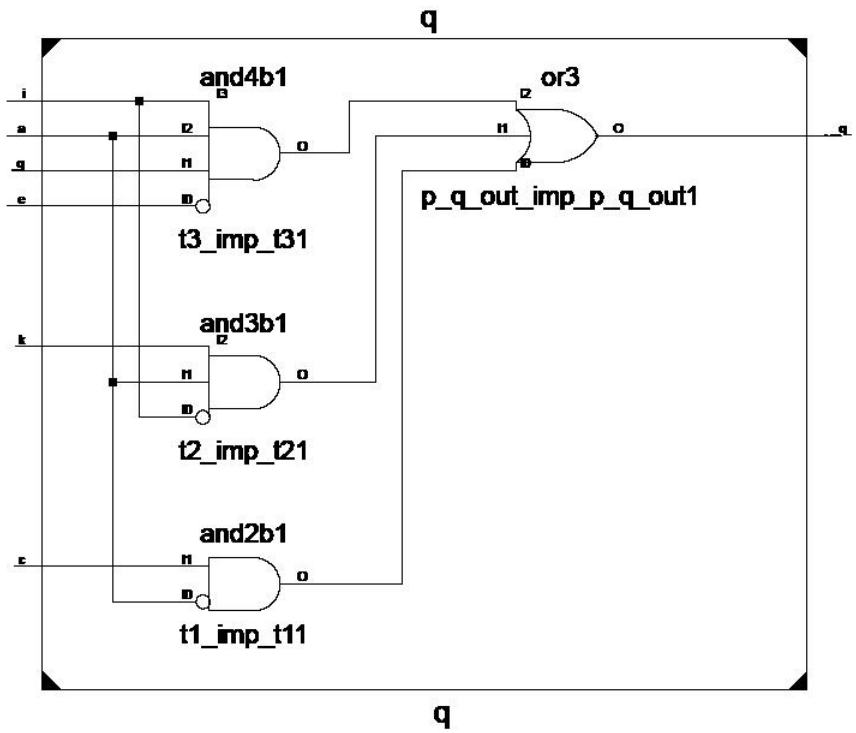

Figure 4.4: Logic module for signal q

4.1.3 Logic module for signal s

a e i	s
000	f
001	f
010	j
100	f
110	0
101	0
011	1
111	1

Figure 4.5: Logic for signal s

Output signal s can be modelled as follows:

t1 <= f and not e and not i;

t2 <= f and not a and not e;

t3 <= j and not a and e and not i;

t4 <= e and i;

s <= t1 or t2 or t3 or t4 ;

Then we have the following logic module as shown in figure 4.6

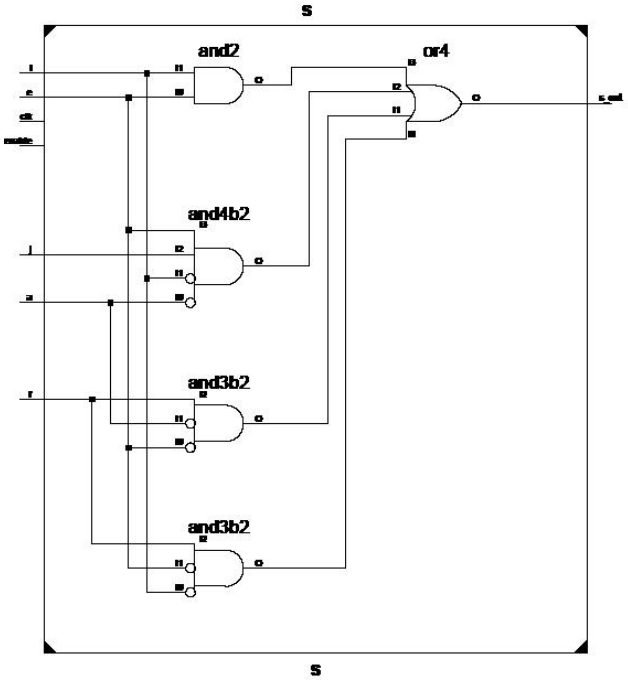

Figure 4.6: Logic module for signal s

4.1.4 Logic module for signal t

a e i	t
000	g
001	g
010	k
100	g
110	0
101	1
011	0
111	1

Figure 4.7: Logic for signal t

Output signal t can be modelled as follows:

t1 <= g and not a and not e;

t2 <= g and not e and not i;

t3 <= k and not a and e and not i;

t4 <= a and i;

t <= t1 or t2 or t3 or t4;

Then we have the following logic module as shown in figure 4.8

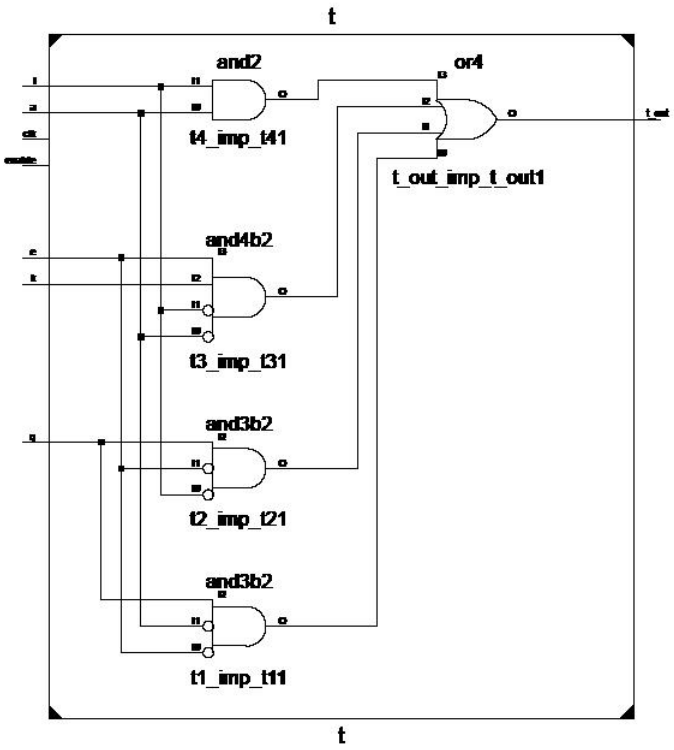

Figure 4.8: Logic module for signal t

4.1.5 Logic module for signal w

a e i	w
000	j
001	0
010	0
100	1
110	1
101	1
011	1
111	1

Figure 4.9: Logic for signal w

Output signal w can be modelled as follows:

t1 <= j and not a and not e and not i;

t2 <= e and i;

w <= t1 or t2 or a;

Then we have the following logic module as shown in figure 4.10

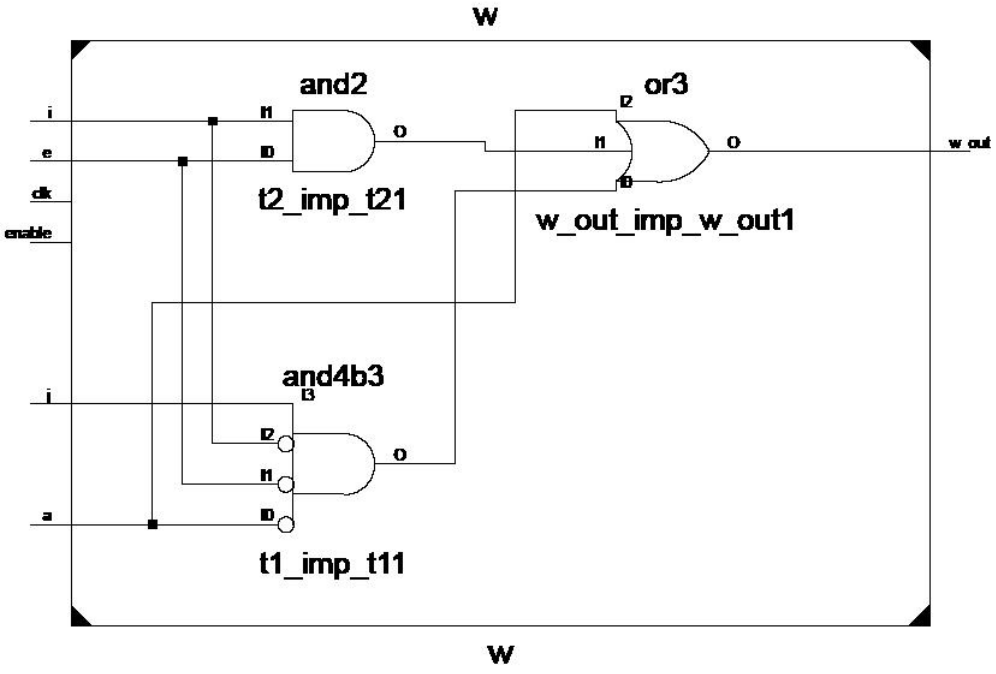

Figure 4.10: Logic module for signal w

4.1.6 Logic module for signal x

a e i	x
000	k
001	0
010	1
100	0
110	1
101	1
011	1
111	1

Figure 4.11: Logic for signal x

Output signal w can be modelled as follows:

t1 <= k and not a and not e and not i;

t2 <= a and i;

x <= t1 or t2 or e;

Then we have the following logic module as shown in figure 4.12

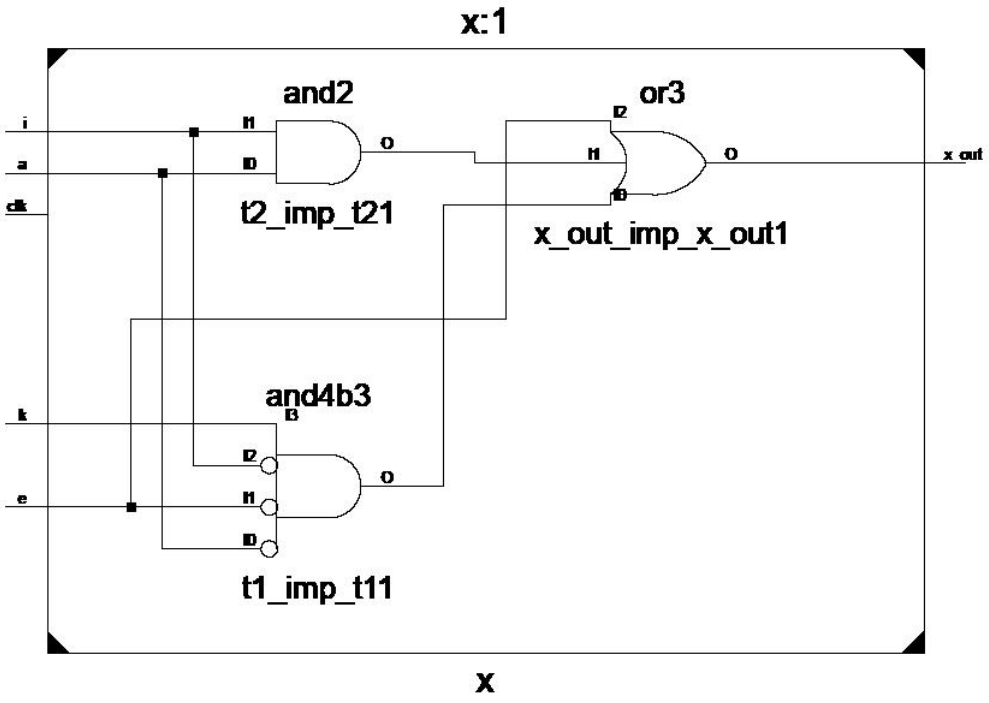

Figure 4.12: Logic module for signal x

4.1.7 Logic module for signal v

Output signal v can be modelled as follows: v <= a or e or i;

4.1.8 Logic for signals r, u, y

Output signal r can be shorted to input signal d. Output signal u can be shorted to input signal h. Output signal y can be shorted to input signal m.

4.1.9 The Compressor Module

All the modules as described previously can be integrated to produce the final compressor module with only 3 logic level gate delays as shown in figure 4.13

4.2 Expander Module Optimization

We observed in Chapter 3 that RTL schematic generated by the VHDL code for the expander logic module has 5 logic gate level delay(including the not gates represented by a circle in the RTL schematic). *Three logic gate delay* module can be achieved by again studying each of the output bits or signals separately.

On analysis similar to that done for the compressor module, we have the following *Three logic gate delay* expander module as shown in figure 4.14

4.3 Simulation

Testbench modules were developed in VHDL for the optimized Compressor and Expansion modules. Inputs as shown in figure 2 were fed to the modules one after another and the corresponding outputs were monitored. Simulation was done using Xilinx ISE 12.1 and ISim 12.1. Results were as in Figure 4.15 and Figure 4.16.

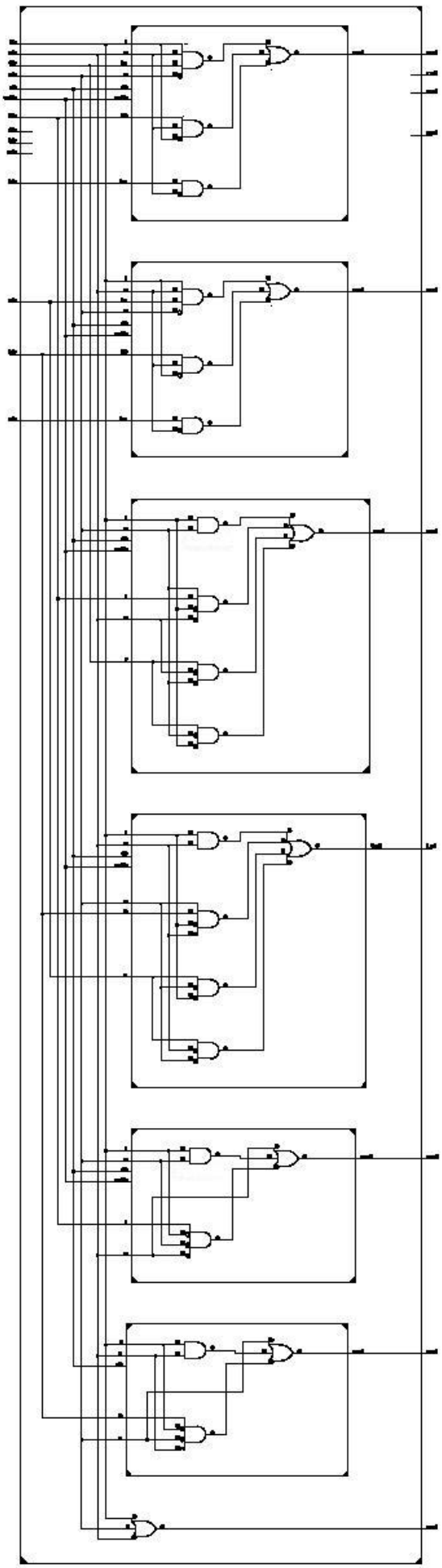

Figure 4.13: The Compressor module

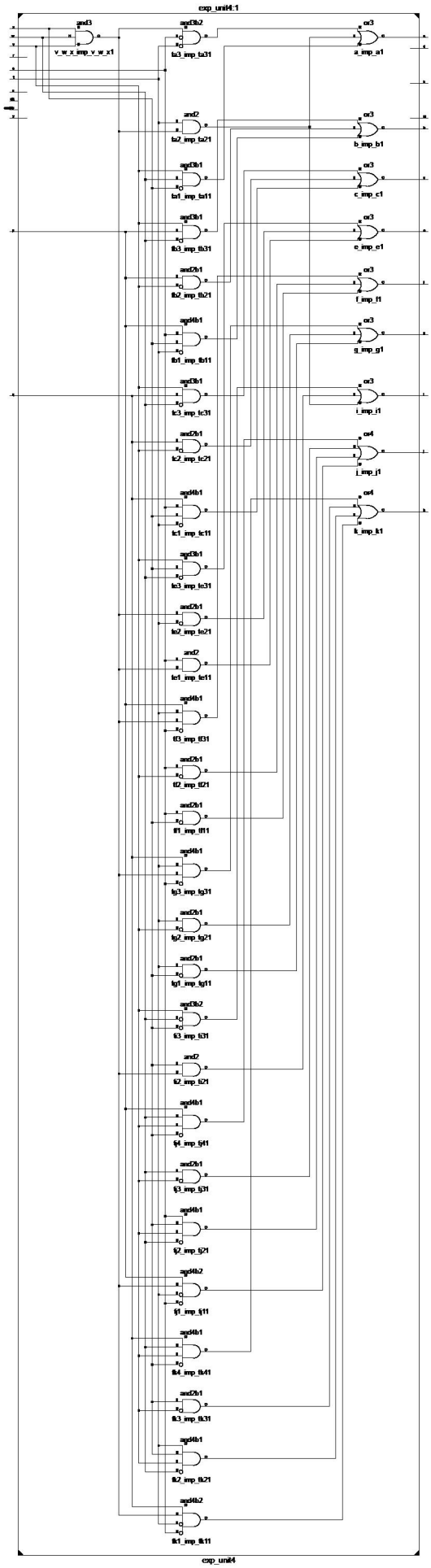

Figure 4.14: The Expander module

Figure 4.15: Compressor module simulation

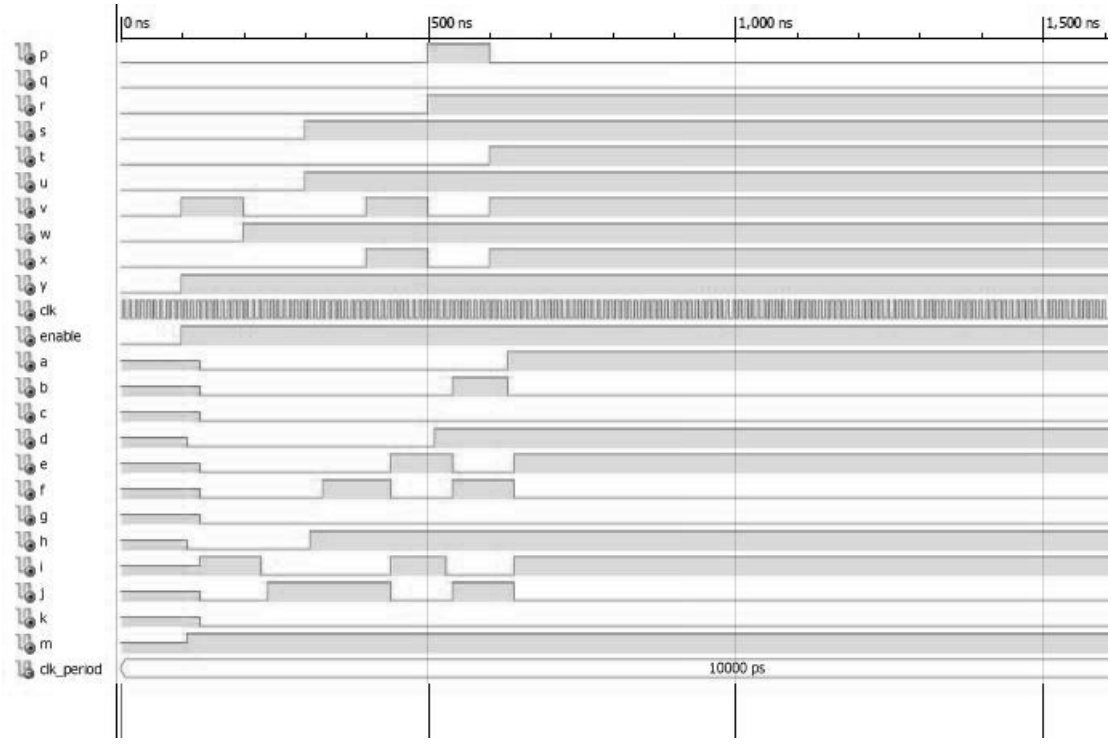

Figure 4.16: Expander module simulation

Chapter 5

Pipelining

In a synchronous pipeline model, clocked high speed latches are used to interface between stages. At the falling edge of the clock pulse, all latches transfer data to the next stages simultaneously [5]. A register stage or latch is referred to be *transparent* when it instantaneously passes data from its input to its output. A latch is referred to be *opaque* when it holds its output constant, regardless of any changes in its input unless or until it receives a clock pulse and becomes transparent [6].

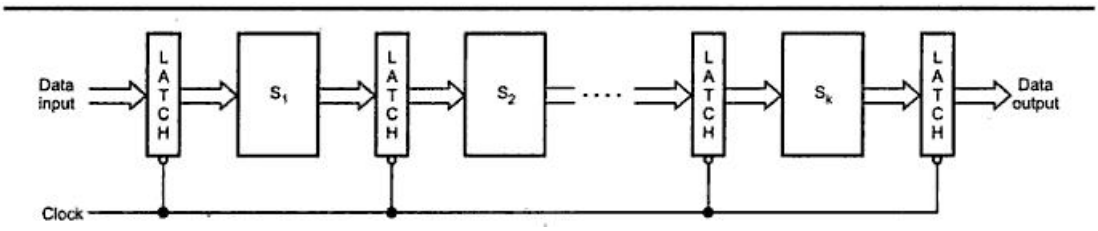

Figure 5.1: Synchronous pipeline model [5]

5.1 Pipeline Stages

The pipeline stages in both compressor and expander modules can be identified as :

1. Not/Precomputation stage
2. And stage
3. Or stage

5.2 Pipelined implementation of compressor and expander module

Latches must be inserted between the itermediate stages in both compressor and expander modules. Some input signals in both the modules are passed directly to the output(d, h, m in compressor module

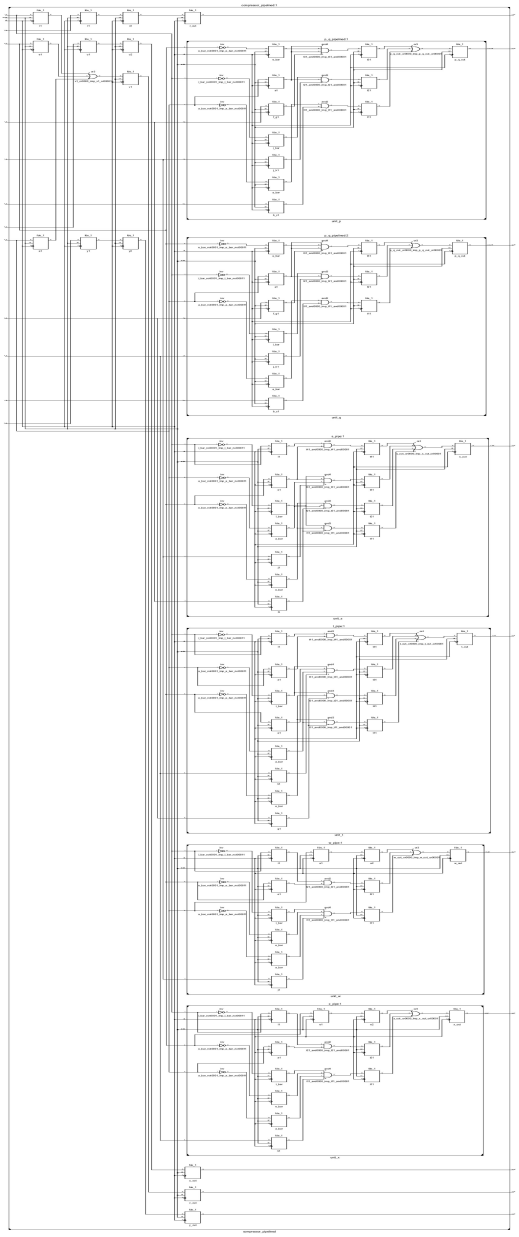

Figure 5.2: RTL schematic for pipelined compressor module

and r, u, y in expander module). For synchronisation, we must insert 3 or 4 latches for each of these signals(according to the design of the other modules) so that they reach the output port in synchronisation with all other signals which require gate level processing. Here we use 4 latches for the above mentioned signals. Then we have the following RTL schematics for compressor and expander module as shown in figure 5.2 and 5.3.

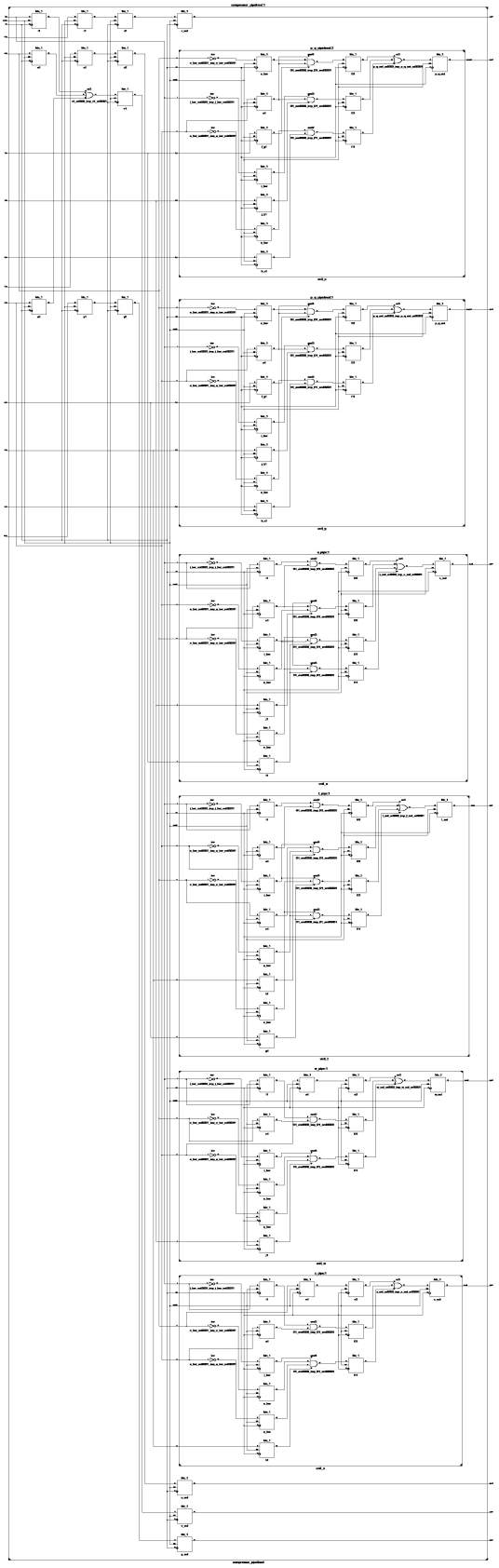

Figure 5.3: RTL schematic for pipelined expander module

Chapter 6

Simulation and Results

We use Xilinx ISE 10.1, Xilinx Spartan 3E XC3S500E, ChipScope Pro 10.1 and ISim 12.1 for simulation and verification of our design.

6.1 Verification of design

ChipScope Pro is a logic analyzer tool which helps in monitoring any signal in a design *dumped* on a Xilinx FPGA board. Signals can be captured in the system and then can be displayed and analyzed using the ChipScope Pro Analyzer tool [9][10][11]. On providing 1001 1001 1001(decimal 999) as input to the compressor module we have the following output in ChipScope Pro as shown in figure 6.1

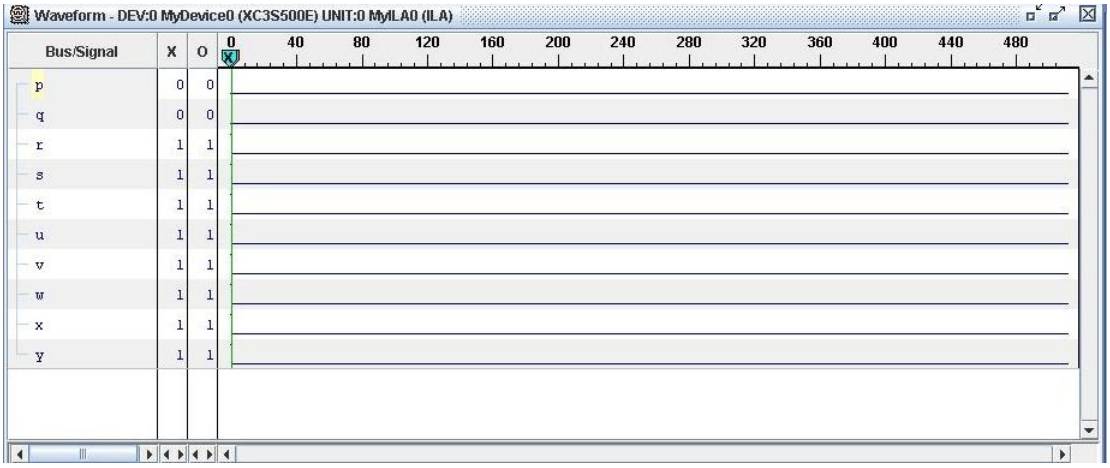

Figure 6.1: ChipScope Pro output for compressor module with input as decimal 999 in BCD format

On providing 001 111 1111(decimal 999) as input to the expansion module we hav the following output in ChipScope Pro as shown in figure 6.2

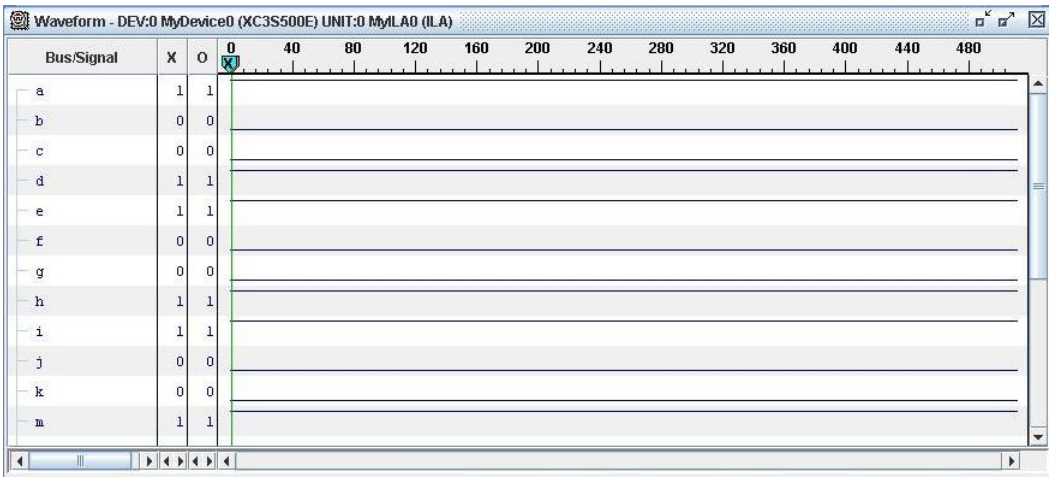

Figure 6.2: ChipScope Pro output for expander module with input as decimal 999 in DPD format

6.2 Simulation and timing analysis

Using ISim 12.1, we can analyse our pipelined design for DPD encoder and decoder. Sequence of inputs were provided to both the modules separately at successive clock cylces. The first output must be generated after 3 negative clock transitions, with input considered to be received at first negative edge. Thereafter, all the outputs must be generated after successive clock cycle (one set of output signals, 10 or 12 bits, after each clock cycle). The timing analysis of the both the modules are as given in figure 6.3 and 6.4.

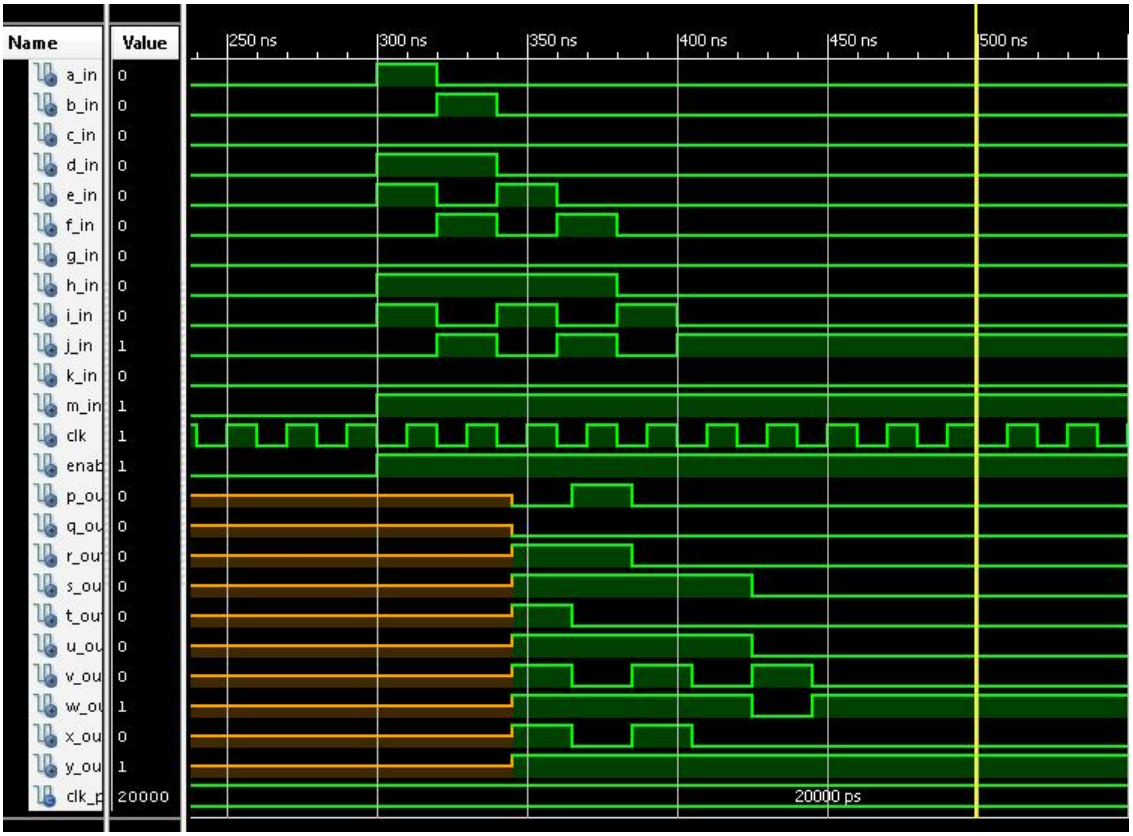

Figure 6.3: Timing analysis for Compressor/Encoder module

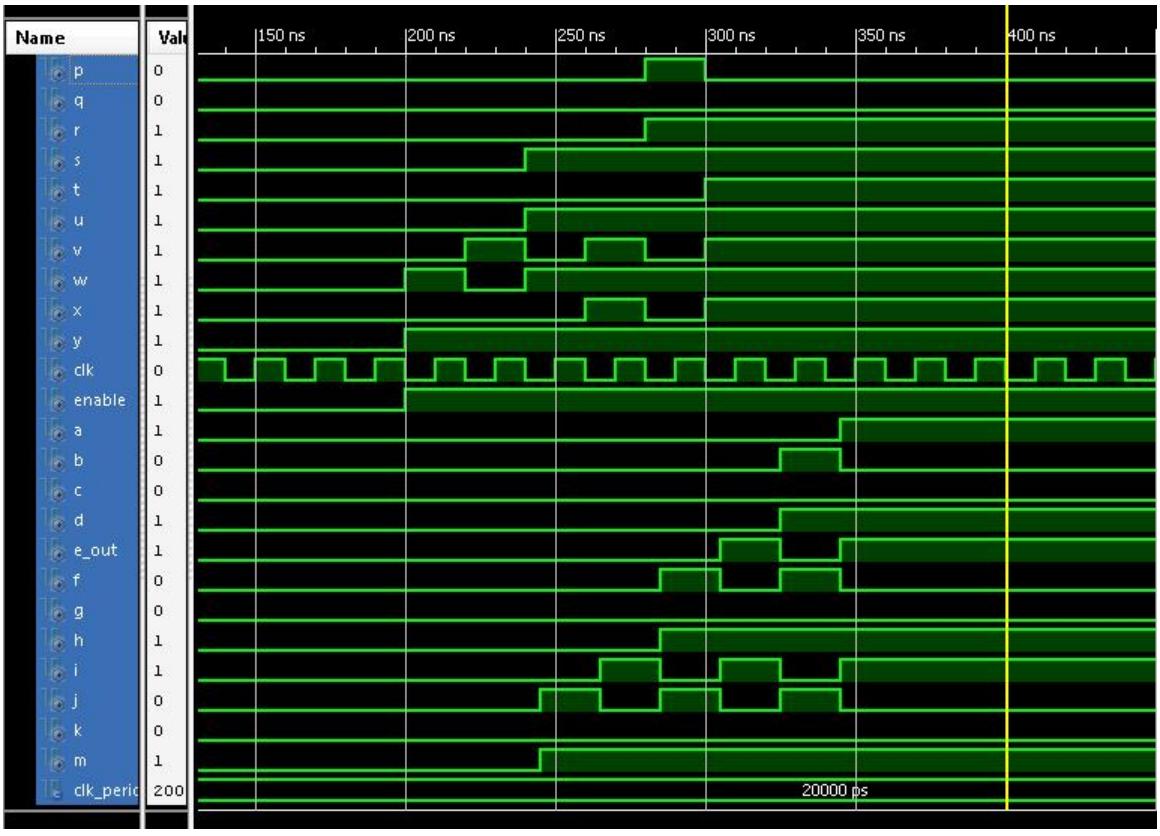

Figure 6.4: Timing analysis for Expander/Decoder module

Chapter 7

Conclusion

If D represents one logic gate delay, then

1. Without Pipelining,Time Per Instruction(TPI) = CPI * CycleTime Here CPI = 1 and CycleTime should be greater than or equal to 3D. So, TPI(non pipelined) = 3D
2. With Pipelining, TPI = D

So, ideal pipeline speedup = 3D/D = 3 = number of stages in pipeline[11].

An ideal speedup is seldom achieved due to the factors like latch overhead and requirement of all the stages to be perfectly balanced etc.

An optimized and pipelined implementation for DPD encoder and decoder was designed, developed, tested and verified on hardware.

Bibliography

[1] Tien Chi Chen and Irving T. Ho. Storage efficient representation of decimal data. *CACM*, 18(1):49 – 52, January 1975.

[2] M. F Cowlishaw. Densely packed decimal encoding. *IEEE Proceedings Computers and Digital Techniques*, 149(3):102 – 104, May 2002.

[3] L Eisen, J. W. Ward, H.W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough. Ibm power6 accelerators: Vmx and dfu. *IBM Journal of Research and Development*, 51(6):1 – 21, November 2007.

[4] Jean Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude Pierre Jeannerod, Lefvre Vincent, Guillaume Melquiond, Nathalie Revol, Damien Stehl, and Serge Torres. *Handbook of Floating Point Arithmetic, 2010*. Springer.

[5] D.A.Godse and A.P.Godse. *Computer Organisation and Architecture*.

[6] M. JACOBSON, Hans. Synchronous pipeline with normally transparent pipeline stages. `http://www.freepatentsonline.com/7076682.html`, 2006.

[7] Michael J. Schulte, Nick Lindberg, and Anitha Laxminarain. Performance evaluation of decimal floating-point arithmetic. `http://domino.research.ibm.com/acas/w3www_acas.nsf/images/conf05/schulte.pdf`, 2008.

[8] M. F Cowlishaw. Decimal floating-point : Algorism for computers. *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, June 2003.

[9] Xilinx. Chipscope pro and the serial i/o toolkit. `http://www.xilinx.com/tools/cspro.htm`, 2011.

[10] Xilinx. Chipscope pro software and cores user guide. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/chipscope_pro_sw_cores_ug029.pdf`, 2011.

[11] www.stanford.edu. Isa implementation basic pipelining. `www.stanford.edu/class/ee282h/handouts/Handout13.pdf`.

[12] `http://www.fpga4fun.com`.

[13] `http://www.tutorial-reports.com/system/files?file=fpga.jpg`.